CS 271 Homework 2

Conditionals and loops in MASM

Due Date: 07/21/19

Michael Payne

TA Signature

1 Introduction

Homework 2 is a program that gets the user's name, and then asks them how many Fibonacci terms they would like displayed on the screen (the program explains that it has to be between 1 and 46).

Homework 2 was designed to help us understand getting and processing string inputs from the user, setting up a post-test loop, and setting up a loop with the MASM loop instruction. It was also supposed to teach us how to keep track of values and validate data.

The basic requirements of the assignment were not difficult. The extra credit was, however, far trickier than I anticipated, and I should have planned this program out with the extra credit (lining up the columns) in mind instead of simply trying to add it to the code that I had already written. If I had done that, I would have probably saved myself a tremendous amount of time.

2 Program Initialization, Internal Register, Definitions, and Constants

"UPPER_LIMIT" is set to 46 and "LOWER_LIMIT" is set to 1 using the equal-sign directive. In the .data section, "space", "fiveSpace", "intro_1", "intro_2", "question_1", "hello", "goodbye", "period", "instructions_1", "instructions_2", and "instructions_3" are all strings that are given a byte for each character. A string named "userName" is assigned 33 bytes (each has the value 0). "count" is assigned 4 bytes and has a value of 1. Finally, "userNum" and "spacingNum" are each assigned 4 bytes, and they both contain uninitialized values.

3 Main Program

Diagram 1 shows all of the main components and the functional logic of the program. The program starts with "Introduce Program", which prints out the title of the program and my name. It then moves on to "Get User Name". This asks for the user's name and then stores the user's response in "userName". It then prints a greeting to the user (which includes the user's name). Next is the "User Instructions" section, which prints out instructions that tell the user that they are going to type in how many Fibonacci terms they want displayed. It then prints out a statement that says that the user may only select between 1 and 46 terms. The next section is "Fibo Number Input", which prints out a prompt telling the user to type in how many Fibonacci terms they want displayed. "Fibo Number Input" has a post-test loop called fiboNumInput that error checks for correct inputs from the user If the user types in an integer within the previously stated range, the input is stored in "userNum". If they type in a number outside the range, the program jumps back to the beginning of the loop and prompts the user to reenter a number. If the user enters a number less than 6, the program jumps to "divSkip" and assigns "spacingNum" a value of one. If the user enters a number greater than 5, the program divides their inputted number by 5 and stores the result in "spacingNum". The program then moves on to "Initializing Fibo Loop", which prints out the number 1. If the user entered inputted a 1 when asked how many Fibonacci terms they wanted to be displayed, the program jumps to "Saying Goodbye" and the program ends. Otherwise, the program uses the value of "spacingNum" as a counter for the loop "initSpacingLoop" repeatedly prints out the value of "space" (which is a space oddly enough) until the counter hits 0. The program then prints out "fiveSpace" and moves onto the "Actual Fibo Loop" section. "fiboLoop"

calculates and prints out Fibonacci terms, and it uses "userNum" as a counter. "fiboLoop" also has the nested conditionals/loops "columnLoop" and "printSpaces", which calculate the appropriate amount of spaces to stick between each term so that all of the terms display as aligned columns (my comments in the code go into a tremendous amount of detail about how this is all done; also, the extra credit section of this report covers this thoroughly as well). After "fiboLoop" runs through those nested loops, it increments "count" (which keeps track of how many terms have been printed). When "count" hits 5, a line break is printed, and "count" is reset to 0. "fiboLoop" then starts over and repeats everything until its counter hits 0. The program then moves onto "Saying Goodbye" where it prints out a message that says goodbye to the user (it uses the user's name in the message).

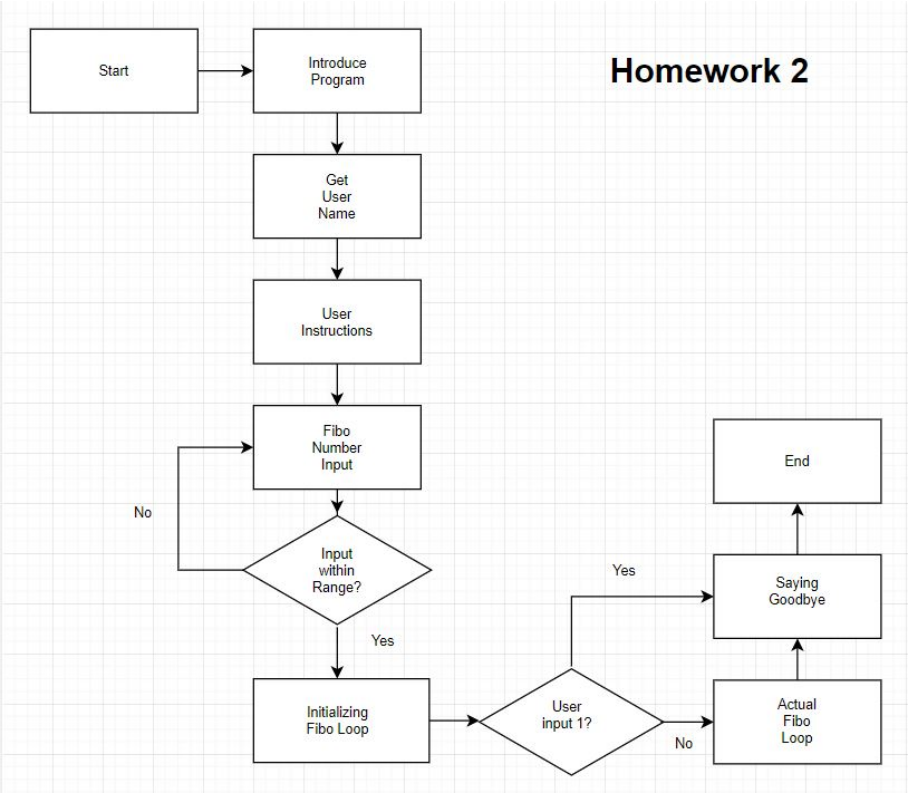

Figure 1: Block Diagram for Homework 2

4 Stored Program Data

1. "Introduce Program" stage
   (a) space = " "
   (b) fiveSpace = " "
   (c) intro_1 = "Fibonacci Numbers"
   (d) intro_2 = "Programmed by Michael Payne"
   (e) question_1 = "What is your name: "
   (f) UserName = 33 DUP(0)

(g) hello = “Hello, ”

(h) goodbye = “Goodbye, ”

(i) period = “”. ”

(j) instructions_1 = “Type in how many Fibonacci numbers you want displayed. ”

(k) instructions_2 = “Enter an integer greater than or equal to 1 and less than or equal to 46.”

(l) instructions_3 = “How many Fibonacci numbers do you want: ”

(m) count = uninitialized value

(n) userNum = uninitialized value

(o) spacingNum = uninitialized value

2. “Get User Name” stage

   (a) userName = “”Mike”

3. “User Instructions” stage

   (a) Nothing Changed

4. “Fibo Number Input” stage

   (a) userNum = 6

   (b) spacingNum = 1

5. “Initializing Fibo Loop” stage

   (a) Nothing Changed

6. “Actual Fibo Loop” stage

   (a) “count” starts at 1 and progressively increments to 5 where it is reset back to 0 (and the process starts up again)

7. “Saying Goodbye” stage

   (a) Nothing Changed

5 Difficulties and Concluding Thoughts

This assignment was fairly straightforward. I struggled a little with constructing the post-test loop that error-checked the user’s input of how many Fibonacci terms they wanted displayed. I was trying to avoid jumping around too much, but I just need to accept that it’s difficult to avoid that while coding in assembly as it is a very restrictive and limited language (at least with my current skill set).

The loop that calculated and posted the Fibonacci terms was tricky in that I couldn’t really figure out a way to incorporate the first 1 (the list of Fibonacci terms starts with 2 1’s) into the loop itself. I had to print it and the subsequent spaces prior to the loop itself.

The hardest part of the loop for me was incorporating code that kept track of the number of terms being printed (which I needed to do in order to insert line breaks at the appropriate points). It was the first time I had ever tried to nest conditionals into a loop, so it took me a little bit of time to figure out exactly where to place everything (my program would often break down towards the end until I ironed out all of the bugs).

Ultimately, what this assignment demonstrated to me is that I really need to spend more time planning and diagramming things out. I have a tendency to simply sit down and start coding without really thinking of how I'm going to approach a problem, and this caused me huge problems when I started tackling the extra credit portion of the assignment (which I will elaborate on in the extra credit section).

6 Extra Credit

The extra credit was what truly bumped up the difficulty of this assignment for me. Initially, I thought that adjusting "fiboLoop" so that it displayed the numbers in aligned columns would be simple, but it actually required me to significantly rework my program, and it ended up consuming a huge amount of time. If I had planned out the program with the extra credit in mind, things might have been easier, but I tried to tack it on at the end, and it simply didn't work out.

I looked at the very last number in the sequence, and it made me realize that the distance between the first digit of any term in a 46-term sequence, and the first digit of the subsequent term (not counting the last term on a line, and the first term on the next line obviously) was always going to be 14 (5 + number of digits to the right of the first digit + number of spaces).

I decided to calculate the number of digits in a term by dividing it by powers of ten until I got a result of 0 (for example, 1 divided by 10 equals 0, which means there's only 1 digit in the term). If you subtract 5 from 14 and then 0 from that, you'd get 9, which would be the number of required spaces for any Fibonacci term (in our 46-term sequence) with only 1 digit.

I wanted the number of spaces to scale with the user's desired number of Fibonacci terms to be displayed, so I decided to use row count to help determine how many spaces to insert (for example, if you had 7 total rows, you'd be trying to fill up a distance of 12 (7 + 5) between the first digit of any Fibonacci term, and the first digit of any subsequent term).

So, figuring out what I wanted to do wasn't tricky. Implementing everything, however, was difficult, because it required me to insert some fairly complicated (at least for me) code into "fiboLoop". It took me forever to produce something that didn't blow up (I did get a lot better at debugging, which was nice), and, again, it would have been much easier, I believe, if I had tried to implement this from the start (when I was planning the program out).

Also, I was actually going to count the number of instructions in my code, but the way I structured my program wouldn't allow me to do that. I didn't think it was feasible to manually count the instructions, so I tried to implement code into my program that counted the instructions for me (I had a variable called instructions that was added to as my the program ran and ran through the various loops). Unfortunately, the additional lines of code made it so the distance between the end of "fiboLoop" and the beginning was too large, and the only way I could think to fix it would be to redo that entire section of the code (I think I would have to turn some parts of "fiboLoop" into procedures that were called). I simply didn't have the time to do that.

7 Source Code

```
TITLE Program Homework2 (Michael_Payne_hw2_code.asm)

; Author: Michael Payne
; Last Modified: 07/21/2019
; OSU email address: paynemi
; Course number/section: cs271
; Project Number: Homework 2 Due Date: 07/21/2019
; Description: This program prints 1 to 46 fibonacci terms. It first displays the
    title of the program and then my name. It then
; gets a number from 1 to 46 (after explaining that it will print fibonacci terms
   ) from the user. If the user doesn't enter a number in
; that range, it prompts the user again until they follow the instructions. The
   program then calculates ands prints out 5 fibonacci terms per line
; and it aligns the terms so that they are in uniform columns (the process on how
    this is done is explained later on in the program). After
; it prints out the fibonacci terms, it says goodbye to the user.

INCLUDE Irvine32.inc
UPPER_LIMIT= 46     ; Maximum number of fibonacci terms that can be printed
LOWER_LIMIT= 1     ; minimum number of fibo terms that can be printed



.data
space   BYTE " ", 0
fiveSpace  BYTE "     ",0
intro_1   BYTE "Fibonacci Numbers", 0
intro_2   BYTE "Programmed by Michael Payne", 0
question_1  BYTE "What is your name: ", 0
userName  BYTE 33 DUP(0)
hello   BYTE "Hello, ",0
goodbye   BYTE "Goodbye, ",0
period   BYTE ". ",0
instructions_1 BYTE "Type in how many Fibonacci numbers you want displayed.", 0
instructions_2 BYTE "Enter an integer greater than or equal to 1 and less than or
    equal to 46.", 0
instructions_3 BYTE "How many Fibonacci numbers do you want: ", 0
count   DWORD 1  ; this is used to keep track of the fibo terms displayed.
userNum   DWORD ?  ; once the user inputs the number of fibo terms he/she wants
   displayed, the result is stored in userNum
spacingNum  DWORD ?  ; userNum is used to determine how many rows of fibo numbers
    will be displayed. the result is stored in spacingNum
```

```
.code
main PROC

; segment 1 - introduce program - This segment introduces the program by listing
   the program title and my name
mov   edx, offset intro_1   ; printing title of program (fibonacci numbers)
call  WriteString
call  CrLf
mov   edx, offset intro_2   ; printing my (programmer's) name
call  WriteString
call  CrLf

; segment 2 - get user name - This segment gets the user's name, stores it in
   userName, and then prints a statement greeting the user.
; (the greeting obviously includes the user's name)
mov   edx, OFFSET question_1  ; printing question asking user's name
call  WriteString
mov   edx, OFFSET userName  ; using readstring to store user's input of name
mov   ecx, 32      ; specifying size of 32 bytes
call  ReadString
mov   edx, offset hello   ; greeting user (greeting includes user's name)
call  WriteString
mov   edx, offset userName
call  WriteString
mov   edx, offset period
call  WriteString
call  CrLf

; segment 3- user instructions - tells user to input how many fibonacci numbers
   they want displayed. then states that user can specify a number in the
; range of 1 to 46
mov   edx, offset instructions_1 ; telling user to get ready to input number of
   desired fibonacci numbers
call  WriteString
call  CrLf
mov   edx, offset instructions_2 ; telling user range of acceptable numbers
call  WriteString
call  CrLf

; segment 4 - Fibo number input - asks user to input desired number of fibo
   numbers. If they give a number outside of acceptable range
; (1 to 46), it rejects input and instructs user to try again. This segment also
   uses the user's input (which is stored in userNum)
; to determine how many rows are ultimately going to be displayed (five terms per
    row as per the assignment instructions). This is
; necessary because it allows the program to display all terms in aligned columns
```

```
    (the number of rows is used in the calculation of how much
; space should be between each term -- this will be elaborated on later). this
   value is stored in spacingNum.
; fiboNumInput is a post-test loop that gets the desired num of displayed terms
   from the user. if the input fails validation, the loop starts
; over and asks the user to input the num again
fiboNumInput:
mov   edx, offset instructions_3 ; asking user to input desired number of
   displayed fibo terms
call  WriteString
call  ReadInt
cmp   eax, UPPER_LIMIT   ; comparing inputted value to constant that has upper
   limit of range
jg   fiboNumInput    ; if larger than upper limit, loop begins again
cmp   eax, LOWER_LIMIT   ; comparing inputted value to constant that has lower
   limit of range
JL   fiboNumInput    ; if less than lower limit, loop begins again
mov   userNum, eax    ; if input passes checks, it is stored in userNum
mov   ebx, 5      ; beginning calculation to determine spacingNum.
cmp   userNum, ebx
JlE   divSkip      ; if num is less than or equal to 5, program jumps to divskip
   (reason why is explained in comments for divskip)
xor   edx, edx     ; clearing edx so division can safely be done
div   ebx       ; dividing user's input by 5
mov   spacingNum, eax    ; storing input into spacingNum
jmp   initFiboLoop    ; skipping divskip and jumping to the initialization of the
    fibo loop
; divskip (it is a part of the fibo number input segment) assigns spacingNum a
   value of 1. This is necessary because any number less than
; or equal to 5 produces a result of 0, which breaks some of nested conditionals
   in the fiboloop section. While this is an awkward fix to
; the problem, it is the easiest thing I could think of doing.
divSkip:
mov   spacingNum, 1    ; assigning value of 1 to spacingNum

; segment 5 - initializing fibo loop - displays the first value (which is a 1) of
    the fibonacci terms. I wasn't clever enough to think of a
; way to get my fibo loop to display the first 1 (fibo terms go like this: 1 - 1
   - 2 - etc.), and the easiest way to get around this was
; to simply display the first 1 outside my fibo loop.
; initFiboLoop exists as a jump point for input values 6 and above from the user
   (for desired number of displayed fibo terms). It prints a
; value of 1 and then copies the value of spacingNum to ecx (so that the
   appropriate amount of between the first two terms can be calculated).
; also, if the user inputs the value 1 for the desired number of displayed terms,
    it is unnecessary to print any spaces or run through any of
```

```
; the subsequent loops. initFiboLoop checks for that, and if it is true, it
   simply skips to goodByeUser and ends the program
initFiboLoop:
mov   eax, 1      ; printing first fibo term. Beginning initialization of fibo
   loop
call  writedec
cmp   userNum, 1     ; checks to see if the user only asked for one term to be
   displayed
je   goodbyeUser     ; if that is the case, the program jumps to the goodbyeuser
   segment and ends the program
mov   ecx, spacingNum    ; copying value of spacingNum to ecx so that proper
   spacing between first two terms can be calculated
; initSpacingLoop calculates the amount of space in between the first two
   fibonacci terms. The formula for spacing is simple.
; It is: (number of rows - (number of digits in term - 1)) + 5
; 5 is the minimum number of spaces between each term (specified by the
   assignment)
; Since the first term is always going to have 1 digit, initSpacingLoop simply
   prints ((value in spacingNum) + 5) number of spaces after
; the first term. initSpacingLoop also sets up eax and ebx for fiboLoop by
   copying 0 to eax and 1 ebx (which allows the program to start
; calculating and printing all of the subsequent fibonacci terms). It then copies
    userNum to ecx (the user's desired number of fibo terms
; is the counter for fiboLoop) and decrements it by 1 since 1 term has just been
   printed
initSpacingLoop:
mov   edx, offset space   ; printing appropriate number of spaces after first
   term
call  writestring
loop  initSpacingLoop
mov   edx, offset fiveSpace  ; fiveSpace adds the required 5 spaces to the spaces
    determined by the number of digits in the term
call  writestring
mov   eax, 0      ; moving 0 to eax to set up fiboLoop
mov   ebx, 1      ; moving 1 to ebx to set up fiboloop
mov   ecx, userNum    ; copying userNum to ecx, so that program can display
   appropriate number of fibo terms
dec   ecx       ; decrementing ecx by 1, because 1 term was just printed
; segment 6 - Actual Fibo loop - Fibo loop calculates and prints numbers from the
    fibonacci sequence. It also has a number of nested
; conditionals that calculate and print the appropriate amount of space between
   each term (the space scales with userNum)
; the Fiboloop also keeps track (well, a nested conditional does) of how many
   terms have been printed. After 5 terms have been printed,
; a line break is printed (for a minimum of 1 line, and a maximum of 10 lines)
; fiboLoop begins by adding ebx to eax (a fibonacci term is found by adding the
```

```
   preceding two terms). the new term (which is in eax)
; is then printed. The values in eax (newly printed fibonacci term), ebx (
   preceding fibonacci term), and ecx (counter for the loop) are
; then pushed onto the stack (so that they can be retrieved later) because ecx,
   ebx, and eax are needed by columnLoop, printSpaces, and fivePrint
; in order to calculate and print the appropriate amount of spaces to print
   before the next fibonacci term is calculated and printed.
; spacingNum (the number of rows) is copied to ecx and the value 10 is copied to
   ebx (columnLoop needs this in order to determine how many
; digits are in the current fibonacci term)
fiboLoop:
add   eax, ebx     ; adding ebx to eax in order to generate new fibonacci term
call  WriteDec
push  ecx       ; pushing ecx, and ebx, and eax to stack in order to store values
    for later use
push  ebx
push  eax
mov   ecx, spacingNum    ; copying row count to ecx so that spaces can be
   calculated
mov   ebx, 10      ; copying 10 to ebx so that spaces can be calculated
; columnLoop begins the work of calculating and printing the appropriate number
   of spaces to print after the fibonacci term in order to display
; the numbers in aligned columns. columnLoop determines the number of digits in a
    number by dividing it by powers of 10. For example, if the
; value in eax is divided by 10, and it produces a 0, that means it has a single
   digit. if it doesn't produce a zero, the value in ebx is multiplied
; by 10, ecx is decremented by 1, and eax has its value restored by copying the
   top of the stack to it. Then the process starts all over again
; until either eax equals 0 or ecx hits 0. When eax is 0 after division, there is
    a jump to printSpaces where the value in ecx is used to calculate
; the number of spaces to print after the current fibonacci term. If ecx hits 0,
   there is a jump to fivePrint (which means that that fibonacci term
; has so many digits that it is only necessary to print the 5 minimum spaces)
   where only the 5 minimum spaces are printed
columnLoop:
xor   edx, edx     ; edx is cleared so that division can safely be done
div   ebx       ; fibonacci term is divided by 10 to determine how many digits it
    has
cmp   eax, 0      ; check results of division to see if it equals 0
je   printSpaces     ; jump to printSpaces where number of spaces are calculated
mov   eax, 10      ; if 0 isn't result, it is necessary to generate next power of
    10
mul   ebx       ; getting next power of 10
xchg  eax, ebx     ; swapping next power of 10 back into ebx
mov   eax, [esp]     ; restoring fibonacci term to eax from stack
dec   ecx       ; decrementing ecx (which contains row amount)
```

```
cmp   ecx, 0      ; if ecx equals 0, jump to fivePrint to print 5 spaces

je   fivePrint
jmp   columnLoop     ; starting loop over
; printSpaces is a loop that uses the value left in ecx to determine how many
   spaces to print. It prints a space and decrements ecx
; until ecx is equal to 0. Once it is finished, it moves on to fivePrint where
   the five minimum spaces are printed.
printSpaces:
mov   edx, offset space   ; printing a space
call  writestring
dec   ecx       ; decrementing ecx until it hits 0 (where the loop ends)
cmp   ecx, 0
jg   PrintSpaces     ; jumping to beginning of loop until ecx equals 0
; fivePrint marks the end of the space printing and calculating loops. It prints
   the 5 minimum spaces and then pops eax, ebx, and ecx off of
; the stack. The values in eax and ebx are swapped so that the newly generated
   fibonacci term is maintained when they are added again when the
; the loop jumps back to the beginning. Count (again, which is used to keep track
    of how many terms have been printed) is incremented. Once
; it hits a value of 5, a line break is printed (so that there are only 5
   fibonacci terms per line)
fivePrint:
mov   edx, offset fiveSpace  ; printing five spaces
call  writestring
pop   eax       ; restoring fibonacci terms to eax and ebx by popping them off of
    the stack
pop   ebx
pop   ecx       ; restoring counter for main fibo loop by popping off of the
   stack
xchg  eax, ebx
inc   count      ; incrementing counter since fibonacci term was printed earlier
cmp   count, 5     ; when counter hits 5, a line break is printed.
jl   done
; lineBreak prints a line brea when count hits 5 and then resets count to 0, so
   that the next line of fibonacci terms can be tracked
lineBreak:
call  CrLf      ; printing line break
mov   count, 0     ; resetting count to 0
; done jumps the program back to the top of the loop so that the next fibonacci
   term can be generated. Once ecx hits 0, the loop ends, and
; the program moves on to the saying goodbye segment
done:
loop  fiboLoop

; segment 7 - saying goodbye - Once the program is finished printing all of the
```

```
   fibonacci terms, it prints a line break, and then says goodbye
; to the user (it uses the user's name when it does this)
goodbyeUser:
call  CrlF      ; printing line break to make sure there is separation between
   the goodbye and the list of fibo terms
mov   edx, offset goodbye   ; printing first part of goodbye emessage
call  WriteString
mov   edx, offset userName  ; printing username
call  WriteString
mov   edx, offset period    ; printing period to end goodbye message
call  WriteString



exit ; exit to operating system
main ENDP


END main
```